Quantitative Design Decisions Measurement using Formal Method

Fangfang Yuan and Kerstin Eder

University of Bristol

July 16, 2009

Project Background

- Design decisions are made to optimize architectures.

Project Background

- Design decisions are made to optimize architectures.
- Methods are available to justify those design options.

Project Background

- Design decisions are made to optimize architectures.
- Methods are available to justify those design options.
- The impact on verification effort has been rarely considered.

Aims and Objectives

- To find a method that allows engineers to estimate the verification effort behind each design option.

University of BRISTOL

Aims and Objectives

- To find a method that allows engineers to estimate the verification effort behind each design option.
- To extract the correlation between some easily obtained metrics from the formal description and the verification effort.

Related works

- Remarkably how little knowledge.

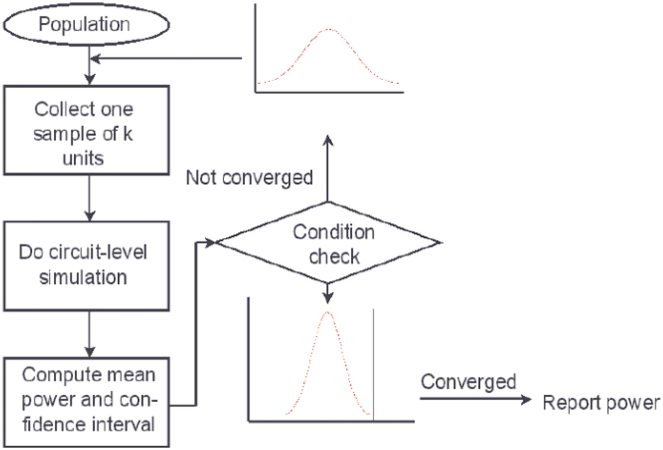
Related works

- Remarkably how little knowledge.
- The methodologies that estimate PCB design effort inspired us.

Related works

- Remarkably how little knowledge.
- The methodologies that estimate PCB design effort inspired us.
- The approaches that lead to the power estimation techniques inspired us.

Power estimation techniques



Reference model requirements

Requirements:

i Extendible

ii Early exploration of design options available

iii Generic

Reference model requirements

Requirements:

i Extendible

ii Early exploration of design options available

iii Generic HOW ?

Design decisions

The design decisions in the reference model:

- Control flow evaluation method

Design decisions

The design decisions in the reference model:

- Control flow evaluation method
- Internal storage access method

Design decisions

The design decisions in the reference model:

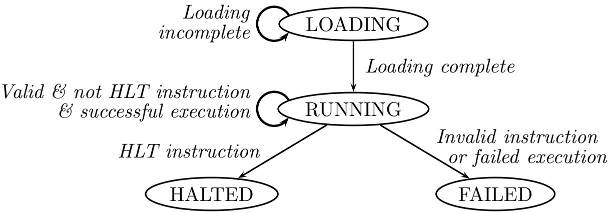
- Control flow evaluation method
- Internal storage access method
- Instruction encoding

Design decisions

The design decisions in the reference model:

- Control flow evaluation method
- Internal storage access method
- Instruction encoding
- Multi-source and multi-destination architecture

The top layer - State Machine



Refinement and event

Refinement

- Step-wise refinement.
- Grouped into distinct layers.

Refinement and event

Refinement

- Step-wise refinement.
- Grouped into distinct layers.

Event

- Horizontally,
  - (a) instructions successfully executed
  - (b) instructions with failed execution
- Vertically,
  - (a) the control flow machine layer
  - (b) the register and the memory machine layer

Refinement - Event split

The opcode of the reference model ISA:

Whole ISA

CmpEq CmpGt CmpLt
CmpFgt CmpFlt
Jmp Branch

LdB LdW LdL SB SW SL . . .
Add Ior Xor Shl Shr . . .
Fadd Fsub Fmul Fdiv . . .

Refinement - Event split

Whole ISA

CmpEq CmpGt CmpLt
CmpFgt CmpFlt
Jmp Branch

LdB LdW LdL SB SW SL ...
Add Ior Xor Shl Shr ...
Fadd Fsub Fmul Fdiv ...

CFM subset

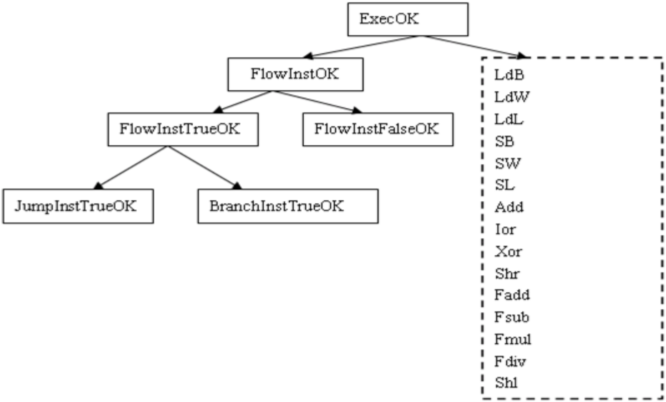
CmpEq CmpGt CmpLt
CmpFgt CmpFlt
Jmp Branch

The rest in the ISA

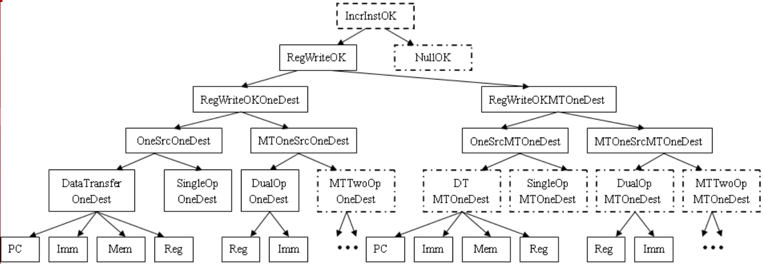
LdB LdW LdL SB SW SL ...
Add Ior Xor Shl Shr ...
Fadd Fsub Fmul Fdiv ...

Refinement - Event split

Refinement - Event split



Refinement - Event split



Refinement - Data refinement

JumpInstTrueOk $\widehat{=}$

FlowInstTrueOk

any

conditionalAvailable
conditional
newPtrAvailable
newInstPtr

where

$grd - inst:\ instArray(instPtr) \in JumpInst$
$grd - status:\ status = RUNNING$
$grd - conditionalAvailable:$
$conditionalAvailable = TRUE$
$grd - conditional:\ conditional = TRUE$
$grd - newPtrAvailable:$
$newPtrAvailable = TRUE$
$grd - newInstPtr:$
$newInstPtr \in InstArrayDom$

then

$act - instPtr:\ instPtr := newInstPtr$

end

Refinement - Data refinement

JumpInstTrueRegOk $\widehat{=}$

JumpInstTrueOk

any

op1Index
op2Index
op3Index
instr
op1Data
op2Data
op3Data
newInstPtr

where

grd — instr :	$instr = instArray(instPtr)$
grd — inst :	$instr \in JumpRegInst$
grd — status :	$status = RUNNING$
grd — op1Index :	$op1Index = Inst2Src0Index(instr)$
grd — op2Index :	$op2Index = Inst2Src1Index(instr)$
grd — op3Index :	$op3Index = Inst2Src2Index(instr)$
grd — op1Data :	$op1Data = regArrayDataLong(op1Index)$
grd — op2Data :	$op2Data = regArrayDataLong(op2Index)$
grd — op3Data :	$op3Data = regArrayDataLong(op3Index)$
grd — newPtrAvailable :	$LongInDom(op3Data \mapsto MaxVector) = TRUE$
grd — newInstPtrAssign :	$newInstPtr = DataLong2Int(op3Data)$
grd — conditional :	$CmpFunc(op1Data \mapsto op2Data) = TRUE$
grd — newInstPtr :	$newInstPtr \in InstArrayDom$

with

conditionalAvailable :	$conditionalAvailable = TRUE$
newPtrAvailable :	$newPtrAvailable = TRUE$
conditional :	$conditional = CmpFunc(op1Data \mapsto op2Data)$

then

act — instPtr :	$instPtr := newInstPtr$

end

Some other tricks

In the *RM* layer, the mapping of the register system is defined as

$$regArray \in RegArrayDom \rightarrow Data.$$

Action in *RegWrite* event

regArray(Index):=srcData

Some other tricks

$$regArray \in RegArrayDom \rightarrow Data.$$

Action in *RegWrite* event

regArray(Index):=srcData

$$\Downarrow$$

Action in *RegWrite* event

regArray:=regArray $\lhd\!\!- \{Index \mapsto srcData\}$

University of BRISTOL

The reference model

Machine Name	Number of Events	Number of Steps	Description
State Machine (*SM*)	8	5	Status and instruction groups that change the status are introduced.
Control Flow Machine (*CFM*)	17	5	PC and instruction types are introduced. *CFM* refines the control flow subset of instructions in *SM*.
Register Machine (*RM*)	42	5	The internal storage is introduced. *RM* refines the register read and write subset of instructions in *CFM*.
Memory Machine (*MM*)	56	7	The external storage is introduced. *MM* refines the memory read and write subset of instructions in *RM*.

The reference model

Design Options in the reference model

- Control flow layer
  (a) Different control flow evaluation options
- Register machine layer
  (a) Different access methods
  (b) Different instruction size and encoding
  (c) Various numbers of sources and destinations available
- Memory machine layer
  (a) Different byte ordering

The metric

The *verification effort* is defined as the number of *tests* that have to be carried out to achieve coverage closure.

The *extra* verification effort brought by a design decision is measured by the *extra* number of events at corresponding level of abstraction.

Case study

CRISP - Cryptographic RISC Processor

The design decisions implemented in CRISP are as follows:

- Compare-and-branch control flow instruction
- Bit aligned control flow instructions
- A set of registers for internal storage
- Fixed instruction encoding
- Up to 4 sources and 2 destinations
- Look up table for logical operations
- Constant registers

Compare-and-branch

	CRISP[a]	Model with condition flag[b]
Number of instructions	$4 \times 2 \times 2^{c}$	$4 + 4$
Number of events	64	$8 + 16$

[a] Non bit-aligned control flow instruction

[b] With the same number of compare functions

[c] $16 = Functions \times \{Jmp, Branch\} \times \{imm, reg\}$

Compare-and-branch

	CRISP[a]	Model with condition flag[b]
Number of instructions	$4 \times 2 \times 2^{c}$	$4 + 4$
Number of events	64	$8 + 16$

If m is the number of compare functions,

[a] Non bit-aligned control flow instruction

[b] With the same number of compare functions

[c] $16 = Functions \times \{Jmp, Branch\} \times \{imm, reg\}$

Compare-and-branch

	CRISP[a]	Model with condition flag[b]
Number of instructions	$4 \times 2 \times 2^c$	$4 + 4$
Number of events	64	$8 + 16$

If m is the number of compare functions,

$$4 \times m = 2 \times m + 16$$

[a] Non bit-aligned control flow instruction

[b] With the same number of compare functions

[c] $16 = Functions \times \{Jmp, Branch\} \times \{imm, reg\}$

Compare-and-branch

	CRISP[a]	Model with condition flag[b]
Number of instructions	$4 \times 2 \times 2^c$	$4 + 4$
Number of events	64	$8 + 16$

If m is the number of compare functions,

$$\begin{aligned} 4 \times m &= 2 \times m + 16 \\ m &= 8 \end{aligned}$$

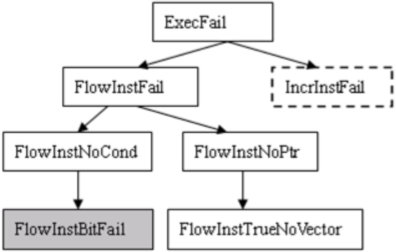
[a] Non bit-aligned control flow instruction

[b] With the same number of compare functions

[c] $16 = Functions \times \{Jmp, Branch\} \times \{imm, reg\}$

Bit-aligned CF instruction

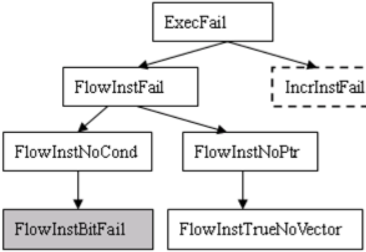
Obviously, there are 4 events per non bit-aligned control flow instruction.



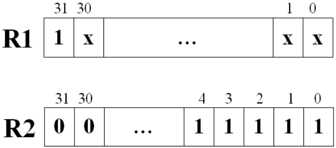
University of BRISTOL

Bit-aligned CF instruction

Obviously, there are 4 events per non bit-aligned control flow instruction.

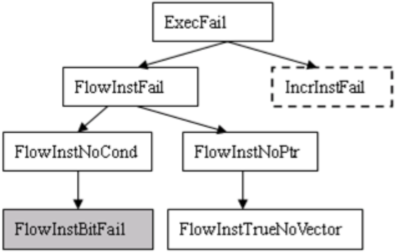


JBIT R1,R2,R3

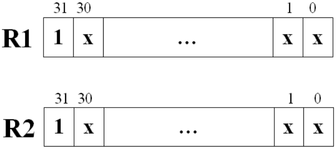


Bit-aligned CF instruction

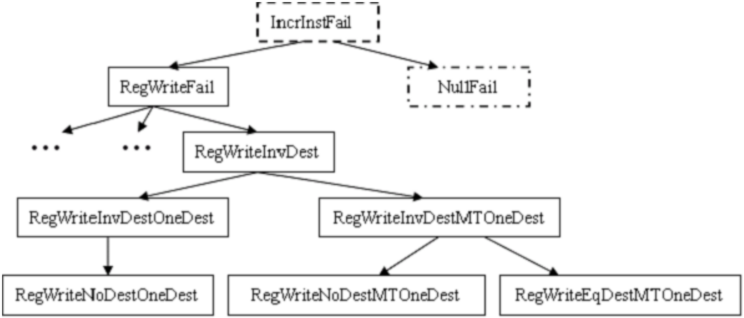
Obviously, there are 4 events per non bit-aligned control flow instruction.



JBIT R1,R2,R3



Multi-destination architecture is more complex. Randomly access register reduces the complexity.



Multi-address random access register architecture

If s is the number of sources, d is the number of destinations, and i is the number of instructions with s operands, the number of events is

$$\begin{cases} i \times (s + d + 3) & \text{if } d = 1 \\ i \times (s + d + 3 + 2^d - 1 - d) & \text{if } d \geq 2 \end{cases}$$

for non-random-access register machine.

$$\begin{cases} i \times 3 & \text{if } d = 1 \\ i \times (3 + 2^d - 1 - d) & \text{if } d \geq 2 \end{cases}$$

for random-access register machine.

Multi-address random access register architecture

If s is the number of sources, d is the number of destinations, and i is the number of instructions with s operands, the number of events is

$$\begin{cases} i \times (s + d + 3) & \text{if } d = 1 \\ i \times (s + d + 3 + C_d^2 + C_d^3 + \cdots + C_d^d) & \text{if } d \geq 2 \end{cases}$$

for non-random-access register machine.

$$\begin{cases} i \times 3 & \text{if } d = 1 \\ i \times (3 + C_d^2 + C_d^3 + \cdots + C_d^d) & \text{if } d \geq 2 \end{cases}$$

for random-access register machine.

Multi-address random access register architecture

If s is the number of sources, d is the number of destinations, and i is the number of instructions with s operands, the number of events is

$$\begin{cases} i \times (s + d + 3) & \text{if } d = 1 \\ i \times (s + d + 3 + C_d^2 + C_d^3 + \cdots + C_d^d) & \text{if } d \geq 2 \end{cases}$$

for non-random-access register machine.

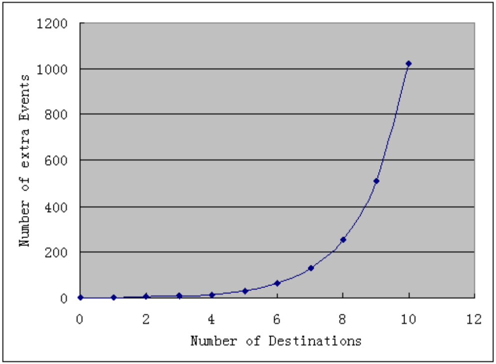
$$\begin{cases} i \times 3 & \text{if } d = 1 \\ i \times (3 + C_d^2 + C_d^3 + \cdots + C_d^d) & \text{if } d \geq 2 \end{cases}$$

for random-access register machine.

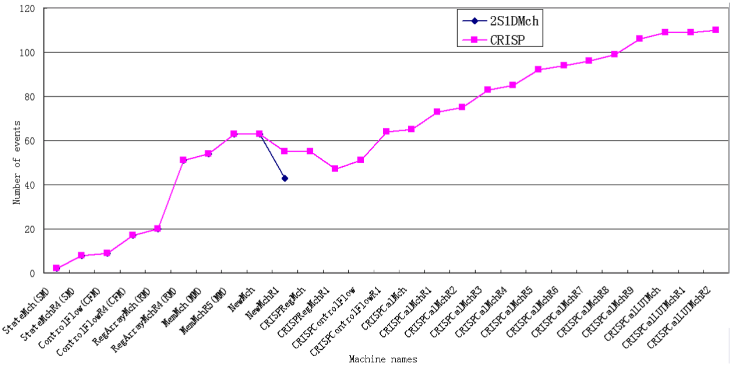
$$\begin{aligned} \because \textstyle\sum_{i=0}^{d} C_d^i = 1 + d + C_d^2 + C_d^3 + \cdots + C_d^d &= 2^d \\ \therefore C_d^2 + C_d^3 + \cdots + C_d^d &= 2^d - 1 - d \end{aligned}$$

Constant register

If d is the number of destinations, and i is the number of instructions, the number of *extra* events needed to cover errors due to writing to constant registers is $i \times (2^d - 1)$.



4S2D vs. 2S1D

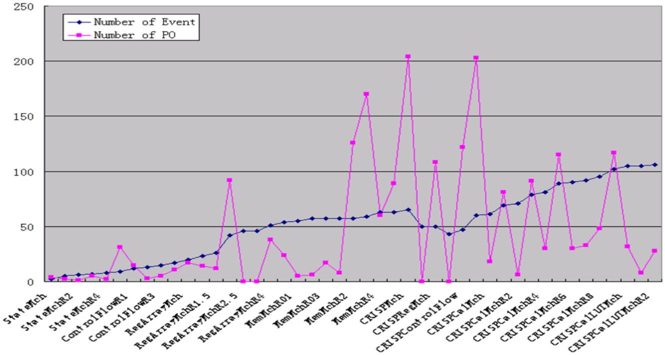


2S1DMch
CRISP
Number of events
120
100
80
60
40
20
0
StateMch(SM)
StateMchR4(SM)
ControlFlow(CFM)
ControlFlowR4(CFM)
RegArrayMch(RM)
RegArrayMchR4(RM)
MemMch(MM)
MemMchR5(MM)
NewMch
NewMchR1
CRISPRegMch
CRISPRegMchR1
CRISPControlFlow
CRISPControlFlowR1
CRISPCalMch
CRISPCalMchR1
CRISPCalMchR2
CRISPCalMchR3
CRISPCalMchR4
CRISPCalMchR5
CRISPCalMchR6
CRISPCalMchR7
CRISPCalMchR8
CRISPCalMchR9
CRISPCalLUTMch
CRISPCalLUTMchR1
CRISPCalLUTMchR2
Machine names

Summary

- A set of random access registers for internal storage (-)
- Fixed instruction encoding (0)
- Compare-and-branch architecture (-8+)
- Up to 4 sources and 2 destinations
  - i Multi-source architecture + random-access registers (0)
  - ii Multi-destination architecture (+)
- Bit aligned instruction (1+)
- Look up table for logical operations (?)
- Constant registers (2^{+})

of Events vs. # of POs



Conclusion

- A method of quantitatively measure the verification complexity using Event-B.
- # of events is better than # of POs

Statistic plug-in

Features:

- Counts metrics.

Statistic plug-in

Features:

- Counts metrics.
- shows the evidence of some types of POs.

Statistic plug-in

Features:

- Counts metrics.
- shows the evidence of some types of POs.
- Outputs to a LaTeX-formatted file.Example.

Statistic plug-in

Features:

- Counts metrics.
- shows the evidence of some types of POs.
- Outputs to a LaTeX-formatted file.Example.

Limitation:

Statistic plug-in

Features:

- Counts metrics.
- shows the evidence of some types of POs.
- Outputs to a LaTeX-formatted file.Example.

Limitation:

- Labels must be hierarchically named.

Statistic plug-in

Features:

- Counts metrics.
- shows the evidence of some types of POs.
- Outputs to a LaTeX-formatted file.Example.

Limitation:

- Labels must be hierarchically named.
- No duplicated event names allowed.

University of BRISTOL

Further development

- POs behind model modifications.
- The tree view plug-in
- Automatic generic model generator
- Etc.

Thank you very much